SESUG Paper 157- 2018

Using a Handful of SAS® Functions to Clean Dirty Data

Ben Cochran, The Bedford Group, Raleigh, NC

ABSTRACT:
A function is a routine that returns a value based on its arguments. Hundreds of functions are available through SAS and they can be used to do a number of tasks. This paper looks specifically at ways that you can use functions to clean data. Functions are also used widely to manipulate data. This paper will look at ways to manipulate dirt right out of the data. After a brief introduction, this paper is divided into six sections: each looking at a main function, but also included will be a discussion as to how to use several functions to get the job (cleaning data) done that allows users to create their own formats. That tool is the FORMAT procedure.

INTRODUCTION:

Functions are very powerful routines that can be used in a number of places in SAS. While this paper looks at using them in the DATA step, it is the author's hope that enough detail is shown so that users can apply their knowledge and skills to see how they can be used in PROC steps as well. (Functions can be used in several procedures, most notably PROC SQL, but they can also be used in WHERE statements in other procedures as well. The typical syntax is**:**

```
function ( argument... )
```

Figure 1.

Arguments**:**

- are generally separated by commas**,**
- can be variables**,**
- can be expressions**,**
- can be constants**,**
- can be other functions**.**

This paper is divided into six sections with each one emphasizing a function. Each section will also have other functions that play minor roles. The six sections are:

1. ELIMINATING LEADING ZEROS:

Sometimes you may have character variables that start with numbers. When these numbers start with leading zeros, some users have found that this may present problems when moving data from SAS to other kinds of data like spreadsheets, for example. The following example shows a DATA step that can remove leading zeros.

```
data leading_0;
  input Number $;
  Non_0 = indexc(Number, '123456789');
  New_Number = substr(Number, Non_0);
cards;
0123
007_0K
00033Y
proc print;
run;
```

Figure 2.

Here a CARDS statement is used primarily so you can see what the data looks like before being read**.** After the INPUT statement, we see our first function: the INDEXC at work**.** The INDEXC function looks at the first argument and returns the starting location of any of the values in the second argument. Notice the second argument contains a string of numbers.... but zero is missing. So, what the INDEXC function will tell us is the starting location of a non-zero and stores that value in a variable called Non_0. The value of Non_0 is used in the second argument of the SUBSTR function. The SUBSTR function returns a substring of the first argument. The second argument is the starting location of the substring. In this case, where the non-zero starts. The PROC PRINT output shows how this works.

Obs	Number	Non_0	New_ Number
1	0123	2	123
2	007_0K	3	7_0K
3	00033Y	4	33Y

Figure 3.

Note: the two assignment statements in the DATA step could have been writen as one, as follows:

```
New_Number = substr ( Number, indexc ( Number, '123456789') );
```

If leading zeros are unwanted, then they are the same as dirty data.

2. THE LENGTH FUNCTION:

Sometimes data comes at us as one long character string and we need to manipulate it. In this case, we want to create two variables from it. In this case, we want to create CITY and STATE from the data shown here.

VIEWTABLE: Work.City

	city_state
1	King and Queen Court House VA
2	Saint Mary of the Woods IN
3	West Palm Beach FL
4	Outer Banks NC

Figure 4.

In this data, we see that some values of CITY contain as many as five 'words'. The value of STATE takes up the last 2 positions of CITY_STATE**.** How do we write a DATA step to split CITY_STATE into CITY and STATE when STATE starts in different positions within CITY?

```
data city_state;
   set city;
   length state $ 5;
   len=length(city_state);
   state = scan(city_state, -1);
   city=substr(city_state, 1, len - 3);
run;
```

Figure 5.

The first assignment statement uses the LENGTH function to get the length of the value of CITY_STATE and stores it in a variable called LEN.

The second assignment statement uses the SCAN function to get the rightmost 'word' from CITY_STATE and stores it in a variable called STATE.

The third assignment statement creates CITY and uses the value of the variable LEN created in the first assignment statement. In this case, a subtraction expression is used as the third argument for the SUBSTR function.

Opening a the viewtable window on the dataset shows the results.

VIEWTABLE: Work.City_state

	city_state	state	len	city
1	King and Queen Court House VA	VA	29	King and Queen Court House
2	Saint Mary of the Woods IN	IN	26	Saint Mary of the Woods
3	West Palm Beach FL	FL	18	West Palm Beach
4	Outer Banks NC	NC	14	Outer Banks

Figure 6.

3. THE TRANSLATE FUNCTION:

Sometimes dirty data comes at us with characters values that should be numbers. The most common examples of this is when a lower case 'L' is found in a numeric value instead of a 1; and the letter 'o' found in a string of digits instead of 0. The STREET_ADDRESS in the data below illustrates this situation.

VIEWTABLE: Sasuser.A_patient

	patient_id	patient	street_address
1	A01101	Ms. Jean Smith	4 Corner St.
2	A99126	Mr. Ronald Moore	130 Market St.
3	B031073	Ms. Beth Adams	442l Glenwood Ave.
4	B001324	Mr. Gus Polinski	18o Cannon Dr.
5	A03121	Mr. Bill Pegg Jr.	10L Cannon Dr.
6	B991401	Mrs. Mary P. Fox	2l0 Ear Ave.
7	A021313	Miss Rose Perez	3O1 Lucky Dr.

Figure 7.

The TRANSLATE function is used in the DATA step below. The typical form of the TRANSLATE function is:

```
translate ( character value , go to,  from )
```

Argument 1 is the character value that will be translated. Argument 2 is the value or list of values that will be created. Argument 3 is the 'from this letter' value. Argument 2 and 3 can be a list of values as seen in the DATA step below.

```
data fix_it (keep=numbers new);;
   set sasuser.A_patient;
   numbers = scan(street_address, 1,  ' ');
   new=translate(numbers,'0011', 'OoLI');
 run;
```

Figure 8.

First, the street numbers are isolated and put into the variable NUMBERS. Then the TRANSLATE function is used to convert any one of these letters: 'Oo' to the digit '0' (zero), and any of these letters: 'LI' to the digit '1'. The next step is to rebuild STREET_ADDRESS with all numbers. The DATA Step below is an expansion of the above DATA step. The statements starting with the arrow are new.

```
data fix_it (drop=numbers new );
   set sasuser.A_patient;
   numbers = scan(street_address, 1,  ' ');
   new=translate(numbers,'0011', 'OoLI');
   space = index(street_address, ' ');
   street_address = trim(new) !! substr(street_address, space);
 run;
```

Figure 9.

Opennng the Viewtable window shows the results of the above DATA Step. named INCOME in the code below.

VIEWTABLE: Work.Fix_it

	patient_id	patient	street_address	space
1	A01101	Ms. Jean Smith	4 Corner St.	2
2	A99126	Mr. Ronald Moore	130 Market St.	4
3	B031073	Ms. Beth Adams	4421 Glenwood Ave.	5
4	B001324	Mr. Gus Polinski	180 Cannon Dr.	4
5	A03121	Mr. Bill Pegg Jr.	101 Cannon Dr.	4
6	B991401	Mrs. Mary P. Fox	210 Ear Ave.	4
7	A021313	Miss Rose Perez	301 Lucky Dr.	4

Figure 10.

4. THE ATTRN FUNCTION WITH THE MODTE ARGUMENT:

The ATTRN function returns information about a numeric attribute of an open SAS data set. The typical syntax is:

```
attrn ( dsid,  attribute - namen )
```

Selected values of ATTRIBUTE-NAME are: any, modte, nobs, nlobs, nvars, etc.

For example, use the ATTRN function to find out how many rows and columns are in a data set.

```
3248  data _null_;
3249       dsid=OPEN('sashelp.class');
3250       if dsid ne 0 then do;
3251          totobs = ATTRN(dsid, "NOBS");  put totobs=;
3252          totvars= ATTRN(dsid, "NVARS"); put totvars=;
3253       end;
3254       rc = CLOSE(dsid);
3255  run;

totobs=19
totvars=5
NOTE: DATA statement used (Total process time):
```

Figure 11.

Many users want to know how old, or fresh, is the data. Use the ATTRN function to see how old the dirt is that is in the data.

```
data _null_;
   dsid=OPEN('SASHELP.CLASS');
   Update_Dt=attrn(dsid,"MODTE");
   rc = close(dsid);
   call symput('Updte', put(Update_Dt, datetime22.));
run;

title "The Class Dataset was Last Updated: &Updte";

proc print data=sashelp.class(obs=7);
run;
```

```
The Class Dataset was Last Updated:   25SEP08:11:34:53

   Obs    Name       Sex    Age    Height    Weight

     1    Alice      F       13      56.5      84.0
     2    Barbara    F       13      65.3      98.0
     3    Carol      F       14      62.8     102.5
     4    Jane       F       12      59.8      84.5
     5    Janet      F       15      62.5     112.5
     6    Joyce      F       11      51.3      50.5
     7    Judy       F       14      64.3      90.0
```

Figure 12 and 13.

5. THE PROPCASE FUNCTION:

The **PROPCASE** function 'shifts' a character value to the proper case. The typical syntax is**:**

```
propcase ( argument  < , delimeter(s) >  )
```

where :

argument is a character variable or expression

delimiter specifies one or more delimiters that are enclosed in quotation marks. The default delimiters are blank, forward slash, hyphen, open parenthesis, period, and tab.

*** Tip:** If you use this argument, then the default delimiters, including the blank, are no longer in effect.

Sometimes data is dirty and is NOT in the correct case. Notice the values of Street_Address in the data below:

patient_ id	patient	street_address	city_cnty_st
B001324	Mr. Gus Polinski	180 Cannon Dr.	Durham, Durham, NC, 27705-2102
A03121	Mr. Bill Pegg Jr	100 cannon Dr.	Durham, Durham, NC, 27705-2102
A982210	Mr. Fred Gold	705 Cannon Drive	Durham, Durham, NC, 27705-2105
A982212	Mr. Ed D. Cox Jr	752 Cannon Drive	Durham, Durham, NC, 27705-2105
C03102	Mrs. Sue Mathis	2L6 Cannon Drive	Durham, Durham, NC, 27705-2102

Figure 14.

Use a DATA step with the PROPCASE and TRANWRD functions to clean the data seen above.

```
data p_sug.cannon_drive_2;
      set p_sug.a_patient;
      if index(upcase(street_address), 'CANNON') > 0;
      street_address = propcase(street_address);
      street_address = tranwrd(street_address, 'Drive', 'Dr.');
run;
proc print data=p_sug.cannon_drive_2;
run;
```

patient_ id	patient	street_address	city_cnty_st
B001324	Mr. Gus Polinski	180 Cannon Dr.	Durham, Durham, NC, 27705-2102
A03121	Mr. Bill Pegg Jr	100 Cannon Dr.	Durham, Durham, NC, 27705-2102
A982210	Mr. Fred Gold	705 Cannon Dr.	Durham, Durham, NC, 27705-2105
A982212	Mr. Ed D. Cox Jr	752 Cannon Dr.	Durham, Durham, NC, 27705-2105
C03102	Mrs. Sue Mathis	216 Cannon Dr.	Durham, Durham, NC, 27705-2102

Figures 15 and 16.

Notice the 'clean' values of STREE_ADDRESS.

6. THE COMPRESS FUNCTION:

The COMPRESS function returns a character string with specified characters removed from the original string. The typical syntax of this function is:

```
compress ( source  < characters >  <, modifier(s) >  )
```

where

source specifies a character constant, variable, or expression from which specified characters will be removed.

characters specifies a character constant, variable, or expression that initializes a list of characters.

modifier by default, the characters in this list are **removed** from the *source* argument. If you specify the K modifier in the third argument, then only the characters in this list are kept in the result. Specifies a character constant, variable, or expression in which each non-blank character modifies the action of the COMPRESS function. Blanks are ignored. The following characters can be used as modifiers:

You can use the COMPRESS function to perform a fuzzy merge. To illustrate how the COMPRESS function works, the DATA steps below shows how to remove periods, spaces, and vowels from the first argument.

```
data names;
   input name $ 1-16;
   check = compress(name, '. aeiou');
   cards;
LL Beane
L.L.Bean
L.L. Bean
LL. Bena
LL Baene
;
proc print;
run;
```

Figure 17.

The output looks like this...

Obs	name	check
1	LL Beane	LLBn
2	L.L.Bean	LLBn
3	L.L. Bean	LLBn
4	LL. Bena	LLBn
5	LL Baene	LLBn

Figure 18.

As the output illustrates, no matter how NAME is spelled, the COMPRESS function creates CHECK with the same spelling across all observations.

CONCLUSION

Functions are very powerful routines that can be used to do a number of things in SAS... including cleaning data.

Many more examples will be given during the presentation of this paper.

The author can be reached at:
Ben Cochran
The Bedford Group
3224 Bedford Avenue
Raleigh, NC 27607
(919) 741-0370
bencochran@nc.rr.com